Laboratorio di

Sicurezza Informatica

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pluggable Authentication Modules

Marco Prandini

Dipartimento di Informatica – Scienza e Ingegneria

Pluggable Authentication Modules (PAM)

- Beyond authentication, PAM provides
  - Account rights management (time, location, ...)
  - Password management
  - User session housekeeping
  - Stacking of modules performing different kind of checks
  - Individual configuration for each program using it

PAM usage

- To make a program PAM-aware
  - The prototype `<security/pam_appl.h>` is included in the source
  - The executable is linked to the pam libraries, which provides the call mechanisms to the abstract security functions
    - the standard libs are `libpam` and `libpam_misc`
    - verify if a program uses them with `ldd`
- The PAM system is configured to implement the abstract functions
  - In a specific way for the caller program
  - By means of loadable modules
- Each PAM module is installed in `/lib/security`

PAM configuration

- Configuration can be
  - In the single file `/etc/pam.conf`
    - Each row is in the format
      `program module-type control-flag module-path arguments`
  - In a file for each program, placed in /etc/pam.d (most common)
    - Each file has the same name of the program and rows in the format
      `module-type control-flag module-path arguments`
- Example (latter format):

```
auth        sufficient    /lib/security/pam_ldap.so
account     sufficient    /lib/security/pam_ldap.so
password    sufficient    /lib/security/pam_ldap.so
session     optional      /lib/security/pam_ldap.so
auth        requisite     pam_unix2.so
auth        required      pam_securetty.so
auth        required      pam_nologin.so
```

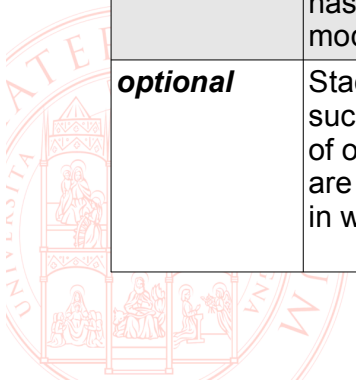
PAM configuration

- *module-type* defines the service to configure. Most programs need at least one line for each possible service:
  - `auth`: authentication service, e.g. password lookup from passwd/shadow
  - `account`: account management functions that aren't related to authentication, e.g.:
    - restrictions based on group membership, time of the day, access path
    - reminders about password expiration
  - `session`: session management, e.g. setup at login and cleanup at logout
  - `password`: handling of authentication tokens update

PAM configuration

- *control-flag* describes how PAM should react to the success or failure of a module

Control flag	Module success result	Module failure result
requisite	Stack execution continues; stack may succeed or fail, depending on outcome of other modules	Stack execution terminates immediately; stack fails
required	Stack execution continues; stack may succeed or fail, depending on outcome of other modules	Stack execution continues, but stack fails
sufficient	Stack execution terminates immediately, with success if no prior ***required*** module has failed, with failure if a prior ***required*** module has failed.	Stack execution continues; stack may succeed or fail, depending on outcome of other modules
optional	Stack execution continues; stack may succeed or fail, depending on outcome of other modules, unless other modules are missing or give inconclusive results, in which case the stack succeeds	Stack execution continues; stack may succeed or fail, depending on outcome of other modules, unless other modules are missing or give inconclusive results, in which case the stack fails



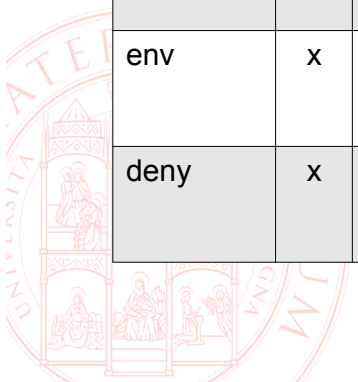
PAM configuration

- *module-path* specifies the absolute path to the shared library that implements the authentication or authorization functions
- after that, it is possible to supply additional arguments that should be passed to the module upon invocation
  - many module-specific parameters
  - some standard arguments, normally supported by all modules
    - debug: enables generation of debugging information either to standard output or via the syslogd daemon.
    - `no_warn`: disables authentication failure logging.
    - `use_first_pass`: instructs the module to use the password entered for the previous module and to return failure if the password does not succeed.
    - `try_first_pass`: instructs the module to attempt to use the password entered for the previous module. If authentication fails, the user should be prompted to enter the password for this module.

Common PAM modules

- A comprehensive documentation can be found at http://www.kernel.org/pub/linux/libs/pam/modules.html
- Most common:

name (filename is pam_*name*.so)	auth	account	session	password	description
unix	x	x	x	x	Implements the traditional Unix authentication, based on /etc/passwd and /etc/shadow files.
time	x				Implements time-based access restriction rules (from /etc/security/time.conf)
nologin	x	x			If /etc/nologin exists, only root is permitted to log in, other users are shown the contents of that file.
env	x				Sets environment variables for the login session, based on the contents of the configuration file (/etc/security/pam_env.conf by default)
deny	x	x	x	x	Always indicates a failure; useful at the end of certain stacks to eliminate the risk of an unauthorized login due to misconfiguration.



Common PAM modules (cont'd)

name (filename is pam_*name*.so)	auth	account	session	password	description
limits			x		Places limits (from /etc/security/limits.conf by default) on users' memory, CPU time, etc. usage
access		x			Uses /etc/security/access.conf (by default) to determine username/machine name pairs that are or aren't granted access.
pwcheck				x	Performs extra checks on password changes, as defined in /etc/login.defs, to improve security on user-selected passwords.
cracklib				x	Checks for passwords that have been used in the past or passwords that are too simple, during password-change interactions.
tally	x	x			Keeps track of the number of times an attempt is made to access an account. It can deny access after a specified number of failures.
warn	x	x	x	x	Allows dumping information to syslog

PAM configuration example 1

- Consider the following configuration:

```
auth        required    /lib/security/pam_unix.so
auth        required    /lib/security/pam_securetty.so
auth        required    /lib/security/pam_nologin.so
```

- How does this work?
  - Any authentication attempt must be approved by all three in order for authentication to succeed.
    - The first PAM module performs standard user and password authentication according to entries in /etc/passwd.
    - The second module, pam_securetty.so, causes a root login to fail unless it is on a terminal listed in /etc/securetty.
    - The final PAM module, pam_nologin.so, results in all logins except root failing if the file `/etc/nologin` exists.
  - Notice the order!
    - Assuming that the file /etc/nologin exists, what users will be able to log onto the system? The answer is that only the root account will be able to log on but only from a secure console.
    - How would this be different if the control flag in the first line was changed from *required* to *sufficient*? Then, root would be able to log in from anywhere, and the /etc/nologin file would have no effect.

PAM configuration example 2

- Consider the following configuration:

```
auth required   pam_unix.so try_first_pass
auth sufficient pam_krb5.so try_first_pass
auth required   pam_env.so
```

Note:
- these modules actually authenticate users
- this one sets environment variables but never returns a failure code

- How does this work?
  - this stack as a whole succeeds if and only if pam_unix.so succeeds;
    - if it fails, its *required* status overrides the *sufficient* status of pam_krb5.so
    - if it succeeds, that success won't be overridden by a failure of the *sufficient* pam_krb5.so
- What happens if the two authentication modules' order is reversed, though?

```
auth sufficient pam_krb5.so try_first_pass
auth required   pam_unix.so try_first_pass
auth required   pam_env.so
```

  - Because the *sufficient* pam_krb5.so module comes first, its success bypasses the later *required* pam_unix.so module, so this stack succeeds if either module succeeds.
- In both cases, though, success of pam_krb5.so bypasses the pam_env.so module, which may not be desirable!

PAM default-deny configuration

- A secure default can be created to block unwanted access by means of any program that hasn't got its own specific PAM configuration
- Create the `/etc/pam.d/other` file and populate it as follows:

```
auth        required    pam_warn.so
auth        required    pam_deny.so
account     required    pam_warn.so
account     required    pam_deny.so
password    required    pam_warn.so
password    required    pam_deny.so
session     required    pam_warn.so
session     required    pam_deny.so
```